https://doi.org/10.3311/PPtr.15837 | 307
Creative Commons Attribution

Periodica Polytechnica Transportation Engineering, 48(4), pp. 307–312, 2020

Fast Prototype Framework for Deep Reinforcement Learning–based Trajectory Planner

Árpád Fehér[1*], Szilárd Aradi[1], Tamás Bécsi[1]

[1] Department of Control for Transportation and Vehicle Systems, Faculty of Transportation Engineering and Vehicle Engineering, Budapest University of Technology and Economics, H-1111 Budapest, Műegyetem rkp. 3., Hungary
[*] Corresponding author, e-mail: feher.arpad@mail.bme.hu

Received: 02 March 2020, Accepted: 11 March 2020, Published online: 29 June 2020

Abstract

Reinforcement Learning, as one of the main approaches of machine learning, has been gaining high popularity in recent years, which also affects the vehicle industry and research focusing on automated driving. However, these techniques, due to their self-training approach, have high computational resource requirements. Their development can be separated into training with simulation, validation through vehicle dynamics software, and real-world tests. However, ensuring portability of the designed algorithms between these levels is difficult. A case study is also given to provide better insight into the development process, in which an online trajectory planner is trained and evaluated in both vehicle simulation and real-world environments.

Keywords

motion planning, reinforcement learning, testing, development framework

1 Introduction

Nowadays, machine learning based solutions gain high importance and are essential components of autonomous vehicle functions. Vehicles with advanced driver assistance systems are equipped with advanced sensor sets and communication solutions to interact with the traffic and the infrastructure. In the development of such solutions, complex simulation environments, and their portability to real-world tests play a more significant role than before (Tettamanti et al., 2018). This potential is made machine learning one of the most intense research fields both for the vehicle industry and related academic institutions. Many research deals with simplified simulation environments, which work great on a theoretical level. However, testing in a simulator with accurate vehicle dynamics and sensor models or real-world conditions raises new issues. This paper deals with a development and test framework for automotive Machine Learning-based solutions. The framework is presented through a reinforcement learning use-case.

Reinforcement Learning (RL) is a powerful subarea of machine learning, though it needs accurate and fast simulation environments. The effectiveness of the deep RL methods, where the underlying agent uses neural networks for action prediction, was first demonstrated in Atari and board games (Mnih et al., 2013). After this breakthrough, many other research fields tried to apply these techniques from which autonomous driving has high importance. Though, contrary to the original finite-state, finite-action approach of the RL problems, most real-world vehicle problems require continuous spaces. Mainly due to the spread of efficient algorithms with continuous output (e.g., Lillicrap et al., 2015), the RL-based solutions in the field of vehicle control are becoming more widespread. Multiple tasks, like car-following (Zhu et al., 2018), lane-keeping (Wolf et al., 2017) lane changing decisions (Hoel et al., 2018) highway merging (Wang and Chan, 2017) or highway maneuvering (Aradi et al., 2018; Nageshrao et al., 2019) has been evaluated by using Reinforcement Learning techniques for automated driving tasks. A concise survey on the topic can be found in (Aradi, 2020).

To solve reinforcement learning problems, a learning agent is placed in an environment where its job is to maximize the cumulative reward from its actions (a_t).

The training process consists of episodes, which generally consist of a series of steps. After each step, the agent gets a reward (r_t), and the environment returns a new state (s_t), see Fig. 1.

Cite this article as: Fehér, Á., Aradi, Sz., Bécsi, T. (2020) "Fast Prototype Framework for Deep Reinforcement Learning–based Trajectory Planner", Periodica Polytechnica Transportation Engineering, 48(4), pp. 307–312. https://doi.org/10.3311/PPtr.15837

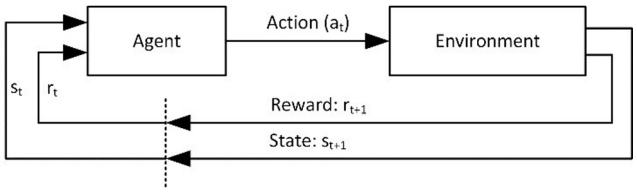

Fig. 1 Reinforcement learning loop

In RL, the simulation environment plays a crucial role in solving real-world control problems. The following main requirements must be met:

- Fast runtime.
- Easy interface with the agent.
- It can be controlled step by step.
- Restartable processes.
- Parametrizable models and close to reality.

AI-based systems are popularly developed in Python, despite its slow interpreter, which became a quasi-standard tool on this field. Many reinforcement learning agents (e.g., Intel Nervana Coach), RL benchmark environments (e.g., OpenAI Gym), and powerful tools (e.g., Tensorflow, Caffe, Keras, PyTorch) are made for Python in recent years.

Also, many open-source automotive simulators support Python on different fields, like TORCS (Wymann et al., 2014) for racing, CARLA (Dosovitskiy et al., 2017) for urban environments, and SUMO (Krajzewicz et al., 2012) for microscopic simulations. However, the portability of the results to real-world scenarios is limited.

On the other hand, detailed commercial vehicle dynamics simulation softwares do not support python development (or direct interface), which raises new issues if we want to test the developed machine learning-based solutions on an automotive industry-standard tool. This article presents the development steps of a framework, where the integration of tools is outlined, from training, through simulation, and finally, real-world testing.

Section 2 presents the developed framework. Section 3 describes the training and test environments of a case study in detail. In Section 4, the results are outlined. Finally, in Section 5, the summary of the experiments and the possible improvements are concluded.

2 Development framework

The developed framework contributes to a fast prototype system for RL vehicle control problems. The framework consists of three development steps, which are presented in Section 2.

Reinforcement learning is a resource-intensive method. A learning process often reaches one million episodes at a complex problem. Many differential equations must be solved to apply a near-real vehicle model at each time step. To achieve the highest possible accuracy and to exclude the possibility of errors, it would be advisable to use a vehicle model of a standard vehicle simulation softwares, e.g. CarSim or CarMaker for this purpose. Besides several technical problems, this would cause a delay because of the interface between the training environment and the software. Hence it is often advisable to use self-implemented models for training. Such a set-up is shown in Fig. 2. An RL agent is placed in a self-implemented environment, which provides a relatively fast learning process that allows the iterative development of RL. The use of classic control solutions can often be useful in an RL environment, which is responsible for control tasks outside RL control (eg.: lane-keeping, cruise control).

Vehicle dynamics simulation softwares are ideal for developing classic control solutions. Among others, they have a very precise, validated, and parameterizable vehicle model. Since most of these softwares do not support Python, it is challenging to use them to develop machine learning solutions. However, interfacing with Python can be solved, making it ideal for testing the learned RL agent. By transferring classical control algorithms from the training environment to simulation software, higher quality requirements can be achieved due to the more efficient development. The Python-based environment model can be expanded with simulation software. Vector virtual CAN network can be used effectively to communicate between the Python and the simulation environment. The simulation tests also show the accuracy of the self-developed vehicle model. Fig. 3 shows this test setup.

The simulation step facilitates real-world vehicle tests, since one can replace the simulation by the real vehicle. Classic control solutions can be run on target hardware responsible for controlling the vehicle, and the RL algorithm can run on a separate, enclosed system. The CAN communication, already developed in the simulation,

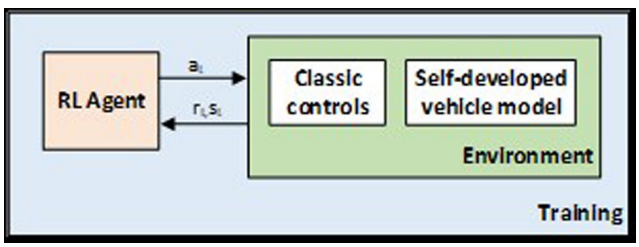

Fig. 2 The reinforcement learning training environment

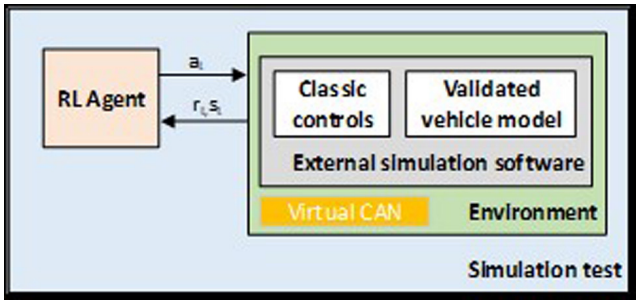

Fig. 3 Testing environment with external vehicle dynamics

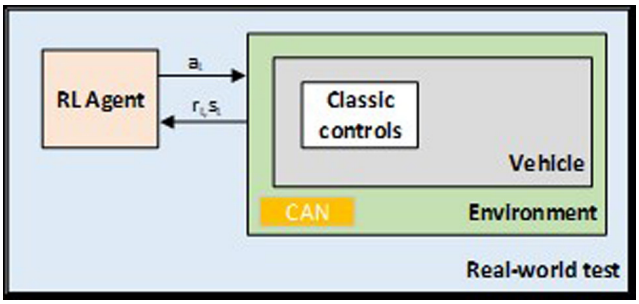

Fig. 4 Environment for real-world testing

establishes a connection between the two devices. Real-world testing can be performed with minimal additional development. The real-world set-up is shown in Fig. 4.

3 A case study

The developed framework is presented through a trajectory planner use case. An earlier version of this RL planner has already been published in (Fehér et al., 2019). Beyond the fundamental problem, the innovations motivated by the developed framework are presented.

3.1 The trajectory planning problem

The Python-based training environment consists of a feasible trajectory generator module, a nonlinear planar single-track vehicle model with a dynamic wheel model, longitudinal and lateral low-level control, and a reward calculation algorithm. It works as a one-step reinforcement learning environment, which also includes a classic control loop.

The inputs of the trajectory planning task are the vehicle state at the start and also the desired end state. Based on this information, a Deep Deterministic Policy Gradient (DDPG) agent determines the intermediate points of the trajectory. The state vector is $[x_s \quad y_s \quad \psi_s \quad v_s]$, where the values are the lateral and longitudinal positions, the yaw, and the speed of the vehicle, respectively. The starting state is fixed to the vehicle position, as given in (Eq. (1)). The final, desired state (Eq. (2), Eq. (3) and Eq. (4)) is an evenly distributed random vector drawn from a set of states that are a bit wider than the feasible targets (Eq. (5)). Too many samples from unfeasible target end-states could lengthen the learning process and hence need to be avoided, though some are beneficial to learn the boundaries.

$$[x_s \quad y_s \quad \psi_s \quad v_s]^T = [0 \quad 0 \quad 0 \quad rand(8.37)]^T \tag{1}$$

$$\begin{bmatrix} x_e \\ y_e \\ \psi_e \\ v_e \end{bmatrix} = \begin{bmatrix} 3v_s \\ rand(-y_{max}, y_{max}) \\ 0.1\psi_{max} + rand(0,1)1.3\psi_{max} \\ v_s \end{bmatrix} \tag{2}$$

$$y_{max} = R_{min} - \sqrt{R_{min}^{\ 2} - x_e^{\ 2}} \tag{3}$$

$$\psi_{max} = -2 * arctan(y_e / x_e) \tag{4}$$

$$R_{min} = 0.1207 v_s^{\ 2.4736} \tag{5}$$

The feasible final state can be determined by an empirical formula (Eq. (5)) as a rule of a thumb, which gives the smallest arc radius that an average vehicle can take at fix speed under normal conditions.

Based on the initial and the end state, the learning agent needs to determine the lateral coordinates of two intermediate points on the trajectory, placed equally between the initial and the endpoint along the longitudinal x coordinate.

A spline is fitted on the four holding points, taking into account the initial and end gradients, which gives the desired trajectory.

Compared to the previously cited paper, the speed during a run is not fixed, but changes between 8 and 37 m/s from episode to episode, hence the complexity of the RL task increased with a new state variable.

A dynamic vehicle model validates the generated trajectory. In order to provide an accurate prediction of the vehicle's behavior at fair computational requirements, a nonlinear planar single track vehicle model containing a dynamic wheel model as well is applied. (Fehér et al., 2019). The model was originally implemented in Python, but even with this time step, the run time was infeasible, considering a large number of iterations in the training process. Because of this, the vehicle model, as well as the solver, was implemented in C, which resulted in a tenfold increase in speed approximately.

To calculate the reward, the vehicle goes along the trajectory using the internal lateral and longitudinal controls. The cumulative sum of the slip, the angular, and distance deviation requirements describe the quality features of the performance of the agent. The episode reward consists of three weighted components (Eq. (6)).

$$R_{episode} = s_w R_{slip} + d_w R_{dist} + a_w R_{angle} \quad (6)$$

The environment defines ten checkpoints distributed on the trajectory. The distance (R_{dist}) and angle (R_{angle}) rewards were calculated at the checkpoints, and the slip reward (R_{slip}) was calculated at all time steps, and (s_w, d_w, a_w) are weighting constants. In the earlier version, it was observed that the trained agent had predicted asymmetric trajectories. It received the same reward for a difficult arc calculated for the last quarter of the course as for a course with evenly distributed difficulty. Therefore, the density of the checkpoint distribution is higher towards the end of the track.

For longitudinal control tasks, a simple PID can adequately handle the problem. The Stanley method (Thrun et al., 2006) is used for lateral control.

3.2 Simulation environment

IPG CarMaker software was used to test the developed RL solution. It is a versatile simulation software for performing vehicle tests in an advanced virtual environment, containing an intelligent driver model, a detailed vehicle model, and highly flexible models for roads and traffic. The software provides C and Simulink interface to its internal models. This allows us to modify or replace those with self-developed control solutions. This feature of CarMaker has been exploited to develop the presented framework.

A Simulink environment was used for the test of the trajectory planner, but the RL environment model is not completely replaced by CarMaker software. The Python environment remains responsible for trajectory design in collaboration with the trained agent to reduce the need for development. Furthermore, it is up to CarMaker to run the vehicle model and the longitudinal and lateral controls. This setup has many benefits: a precise validated vehicle model can be used, and classic control solutions can be effectively run and further developed in a Simulink environment. The Python environment transmits the inputs of the lateral and longitudinal controls and receives the position of the vehicle via Virtual CAN.

Simulink provided a chance to replace the poorly performing lateral Stanley controller to Model Predictive Control (MPC). The Lane Keeping Assist Simulink block decreases the lateral deviation and relative yaw angle, calculates optimal steering angle while providing constraints using adaptive MPC. The prediction model, in this case, is a dynamic single-track model.

The accurate 3D environmental model of the ZalaZone automotive proving ground (see Fig. 5) (Szalay et al., 2019) and used the IPG CarMaker model of it (BME Automated Drive Lab, 2020) for testing purposes.

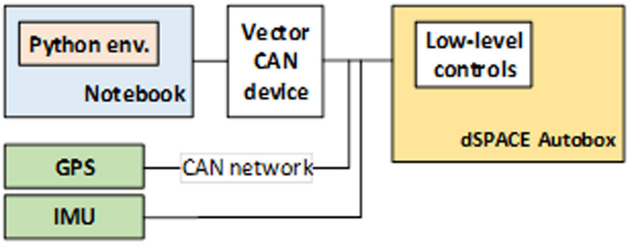

Fig. 5 IPG CarMaker model of the ZalaZone test track

3.3 Vehicle side implementation

The simulation environment is designed so that the controls used there can be ported to a real vehicle with slight effort. Tests can be easily performed on the

ZalaZone with GPS localization. As Fig. 6 shows, Vehicle control solutions are implemented by dSPACE Autobox hardware, which can communicate with the Python environment via its CAN interface. An in-vehicle IMU serves to evaluate the performance of the trajectory planner.

4 Results

Evaluation of a trained system consisting many iterations, is an inherent part of reinforcing learning development. By drawing conclusions, the reward system is refined, weights are re-parameterized, and environmental elements are fine-tuned for better results in each iteration. The result of the presented research is an effective developer and test framework that has contributed to the development of a trajectory planner. Figs. 7 and 8 show examples of the trajectory design development process using this presented framework.

During the training, the evaluation phase runs and computes the reward at the end of the trajectory. When performing overtaking maneuvers, by the end of the trajectory, the distance and angle error is increased, which did not decrease the overall reward much, but greatly influences the errors in the straight section after the trajectory. CarMaker software provides the ability to place virtual sensors on the vehicle, which is contributed to the discovery of this issue. Fig. 7 shows the lateral acceleration of the inertial sensor placed on the vehicle. The blue line indicates the acceleration curve before the enhancement. It can be seen large accelerations at the end of the trajectory and after that on the straight line.

The red line shows the result of the training after modifying the reward system. The absolute value of accelerations is lower, and its distribution is more balanced. Fig. 8 shows the planned trajectories (a blue line represents the original trajectory and a red line shows the result after enhancement). It seems that after the latter trajectory is much more symmetrical, which leads to better lateral accelerations.

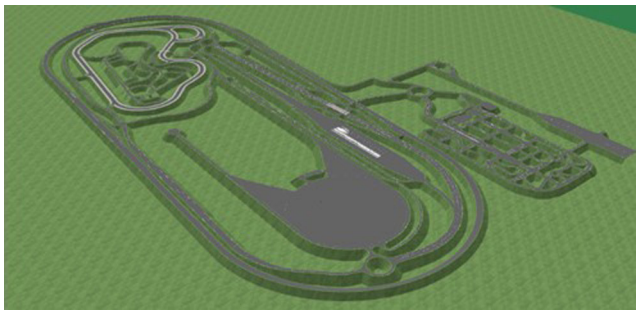

Fig. 6 Actual testing environment for test track evaluations

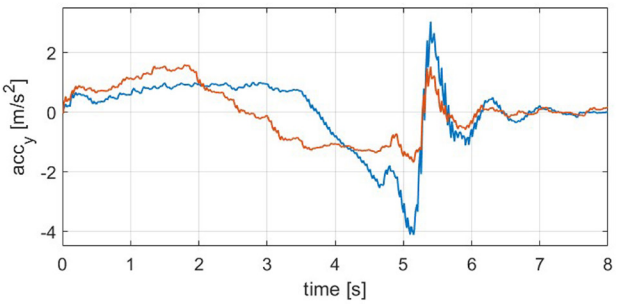

Fig. 7 Lateral accelerations before and after development

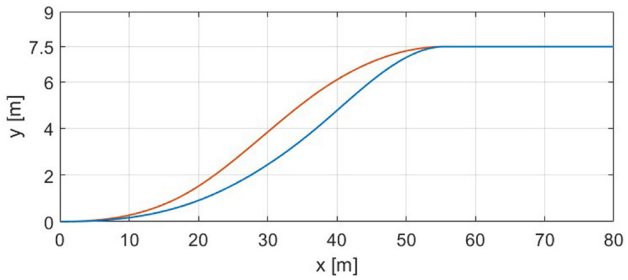

Fig. 8 Planned trajectories before and after development

The test revealed that the performance of the Stanely controller in the learning environment is poor in a more detailed environment. CarMaker provides detailed steering mechanics and the actuator dynamics which is handled by an MPC controller for better performance. The simulation also showed that despite the same parameters, the vehicle model used for training is different from an industry-standard CarMaker model. Further improvements are needed for more accurate operation.

5 Conclusion

Considering the requirements, it is worth to separate the training phase and the validation of the results. In many cases, the environmental model, with its own implementation, leads to the best solution in the training phase, which is validated with an industry-standard simulator software when evaluating the results.

Running in the simulator has shown that steering actuator-dynamics cannot be ignored during controller design.

The advantage comes from the fact, that the developed solution can be tested with high efficiency in an office environment, giving a better chance of success proving ground tests.

Acknowledgment

The research reported in this paper was supported by the Higher Education Excellence Program in the frame of Artificial Intelligence research area of Budapest University of Technology and Economics (BME FIKP-MI/FM).

References

Aradi, S. (2020) "Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles", e-Print archive, arXiv:2001.11231, Ithaca, New York, NY, USA, Cornell University, [online] Available at: http://arxiv.org/abs/2001.11231 [Accessed: 05 March 2020]

Aradi, S., Becsi, T., Gaspar, P. (2018) "Policy Gradient Based Reinforcement Learning Approach for Autonomous Highway Driving", In: 2018 IEEE Conference on Control Technology and Applications (CCTA), Copenhagen, Denmark, pp. 670–675. https://doi.org/10.1109/CCTA.2018.8511514

BME Automated Drive Lab (2020) "Models of the ZalaZONE automotive proving ground in different file formats for simulation software", [online] Available at: https://github.com/BMEAutomatedDrive/ZalaZONE-automotive-proving-ground-virtual-simulation-models [Accessed: 07 March 2020]

Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V. (2017) "CARLA: An Open Urban Driving Simulator", In: Proceedings of the 1st Annual Conference on Robot Learning, Cambridge, MA, USA, pp. 1–17.

Fehér, Á., Aradi, S., Hegedűs, F., Bécsi, T., Gáspár, P. (2019) "Hybrid DDPG Approach for Vehicle Motion Planning", In: Proceedings of the 16th International Conference on Informatics in Control, Automation and Robotics, Prague, Czech Republic, pp. 422–429. https://doi.org/10.5220/0007955504220429

Hoel, C. J., Wolff, K., Laine, L. (2018) "Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning", In: 2018 21st International Conference on Intelligent Transportation Systems (ITSC), Maui, HI, USA, pp. 2148–2155. https://doi.org/10.1109/ITSC.2018.8569568

Krajzewicz, D., Erdmann, J., Behrisch, M., Bieker-Walz, L. (2012) "Recent Development and Applications of SUMO - Simulation of Urban MObility", International Journal On Advances in Systems and Measurements, 5(3), pp. 128–138.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wiersta, D. (2015) "Continuous control with deep reinforcement learning", e-Print archive, arXiv:1509.02971, Ithaca, New York, NY, USA, Cornell University, [online] Available at: http://arxiv.org/abs/1509.02971 [Accessed: 05 March 2020]

Mnih, V. Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M. (2013) "Playing Atari with Deep Reinforcement Learning", e-Print archive, arXiv:1312.5602, Ithaca, New York, NY, USA, Cornell University, [online] Available at: http://arxiv.org/abs/1312.5602 [Accessed:05 March 2020]

Nageshrao, S., Tseng, H. E., Filev, D. (2019) "Autonomous Highway Driving using Deep Reinforcement Learning", In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), Bari, Italy, pp. 2326–2331. https://doi.org/10.1109/SMC.2019.8914621

Szalay, Z., Hamar, Z., Nyerges, Á. (2019) "Novel design concept for an automotive proving ground supporting multilevel CAV development", International Journal of Vehicle Design, 80(1), pp. 1–22. https://doi.org/10.1504/IJVD.2019.105061

Tettamanti, T., Szalai, M., Vass, S., Tihanyi, V. (2018) "Vehicle-In-the-Loop Test Environment for Autonomous Driving with Microscopic Traffic Simulation", In: 2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES), Madrid, Spain, pp. 1–6. https://doi.org/10.1109/ICVES.2018.8519486

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., Lau, K., Oakley, C., Palatucci, M., Pratt, V., Stang, P., Strohband, S., Dupont, C., Jendrossek, L. E., Koelen, C., Markey, C., Rummel, C., van Niekerk, J., Jensen, E., Alessandrini, P., Bradski, G., Davies, B., Ettinger, S., Kaehler, A., Nefian, A., Mahoney, P. (2006) "Stanley: The robot that won the DARPA Grand Challenge", Journal of Field Robotics, 23(9), pp. 661–692. https://doi.org/10.1002/rob.20147

Wang, P., Chan, C. Y. (2017) "Formulation of deep reinforcement learning architecture toward autonomous driving for on-ramp merge", In: 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Yokohama, Japan, pp. 1–6. https://doi.org/10.1109/ITSC.2017.8317735

Wolf, P., Hubschneider, C., Weber, M., Bauer, A., Härtl, J., Dürr, F., Zöllner, J. M. (2017) "Learning how to drive in a real world simulation with deep Q-Networks", In: 2017 IEEE Intelligent Vehicles Symposium (IV), Los Angeles, CA, USA, pp. 244–250. https://doi.org/10.1109/IVS.2017.7995727

Wymann, B., Christos, D., Andrew, S., Eric, E., Christophe, G. (2014) "TORCS: The Open Racing Car Simulator", [online] Available at: http://www.torcs.org [Accessed: 05 March 2020]

Zhu, M., Wang, X., Wang, Y. (2018) "Human-like autonomous car-following model with deep reinforcement learning", Transportation Research Part C: Emerging Technologies, 97, pp. 348–368. https://doi.org/10.1016/j.trc.2018.10.024